SISTEMI OPERATIVI

ESERCIZIO N. 1 del 9 NOVEMBRE 2001

Un sistema di news dipartimentale è costituito da un **news server** che gestisce **G** gruppi di news. Il server può essere acceduto da 2 tipi di applicazioni: **news client** e **G demoni di aggiornamento** da altri server (uno per gruppo). Per motivi di banda, al massimo **N** news client possono accedere al server contemporaneamente. Per motivi di consistenza, quando sta accedendo un demone sincronizzatore nessun news client può accedere.
Si implementi una soluzione usando il costrutto monitor per modellare il **news server** e i processi per modellare i **news client** e i **demoni di aggiornamento** e si descriva la sincronizzazione tra i processi. Nella soluzione si massimizzi l'utilizzo delle risorse. Si discuta se la soluzione proposta può presentare starvation e in caso positivo per quali processi, e si propongano modifiche e/o aggiunte per evitare starvation.

```
program News

const     N = … { numero massimo di client }
type      tipo = (client, demone);  { tipo di applicazione }

type client = process
begin
      repeat
            server.entra(client);
            <legge>
            server.esci(client);
      until false
end

type demone = process(g: integer)
begin
      repeat
            server.entra(demone);
            <aggiorna il gruppo g>
            server.esci(demone);
      until false
end

type news_server = monitor

{ variabili del monitor }
var  nclient : integer;
      { numero di client che accedono }
      ndemoni : integer;
      { numero di demoni che accedono }
      coda : array[tipo] of condition;
      { code su cui sospendere le applicazioni in attesa }
```

```
procedure entry entra (t: tipo)
begin
      if t = client
      begin
            { se ci sono giàN client o c'è un demone }
            if nclient = N or ndemoni > 0 then
                  coda[t].wait;
            { occupo la risorsa }
            nclient++;
      end
      else { t = demone }
      begin
            { se ci sono dei client }
            if nclient > 0 then
                  coda[t].wait;
            { occupo la risorsa }
            ndemoni++;
      end
end

procedure entry esci (t: tipo)
begin
      if t = client
      begin
            { rilascio la risorsa }
            nclient--;
            { se sono l'ultimo client sveglio tutti i demoni }
            if nclient = 0 then
                  while coda[demone].queue do
                        coda[demone].signal;
            else
                  { altrimenti risveglio 1 client }
                  coda[client].signal;
      end
```

```
        else { t = demone }
        begin
            { rilascio la risorsa }
            ndemoni--;
            { se sono l'ultimo demone sveglio N client }
            if ndemoni = 0 then
                for i := 1 to N do
                    coda[client].signal;
        end
end

begin { inizializzazione delle variabili }
        nclient := 0;
        ndemoni := 0;
end

var  server: news_server; { il nostro monitor }
        c1, c2, ... : client;
        d1, d2, ..., dG : demone;

begin end.
```

Starvation

La soluzione proposta presenta starvation nel caso un tipo di applicazione non lasci mai entrare l'altro tipo.

Si può risolvere imponendo un contatore per ogni tipo di applicazione, alternando la priorità ogni tot di accessi consecutivi dello stesso tipo.